

Real-Time Task for Embedded Control Systems

Atef Gharbi, Mohamed Khalgui, Samir Ben Ahmed

LISI, INSAT, Tunis, Tunisia

ITIA, CNR Institute, Italy

FST, University of El Manar, Tunis, Tunisia

Abstract— This paper deals with the study of Real-Time Tasks for Embedded Control Systems following component-based technology. A real-time task is assumed to be a set of components having some properties independently from any real-time operating system. We apply the Priority Ceiling Protocol (PCP) as a method to ensure the scheduling between periodic tasks with precedence and mutual exclusion constraints. We simulate the scheduling of Real-time tasks with PCP through the Cheddar tool.

Keywords- Real-Time Tasks; Embedded Control System; Simulation; Priority Ceiling Protocol (PCP)

I. Introduction

Nowadays, the time to market the embedded systems is becoming shorter and the cost of development is being cheaper. In order to realize this goal (i.e. reducing time and cost), the designers of embedded system propose to reuse the already developed software components. The main benefits of component are: (i) the separation of concerns which means that each component can be defined alone after that all the system can be reproduced; (ii) the modification of component is very simple (such as modification of data, algorithm, connection, ...); (iii) the adaptation of component to any modification; (iv) there use of component (it is not specific for only one application).

However, there are few kinds of component-based technologies (such as Koala [1], PBO [2], PECOS [3], ...) used in the development of embedded control system due to extra-functional properties to be verified (for example quality of service, timeliness, ... [4]). Anyway, each component-based technology has its benefits and its drawbacks.

Although component technologies deal successfully with functional attributes, they provide no support for managing extra-functional properties of systems (such as synchronization, memory optimization, power consumption, and temporal attributes) that cannot be encapsulated in one component with well-defined interfaces as they crosscut the structure of the overall system. To do so, we define at the operational level some sequential program units called real-time tasks. Thus, we define a real-time task as a set of components having some real-time constraints. We characterize a task by a set of properties independently from any Real Time Operating System (RTOS).

We study in particular the scheduling of tasks through a Real Time Operating System. We apply the priority ceiling protocol proposed by [5] to avoid the problem of priority inversion as well as the deadlock between the different tasks. The priority ceiling protocol supposes that each semaphore is assigned a priority ceiling which is equal to the highest priority task using this semaphore. Any task is only allowed to enter its critical section if its assigned priority is higher than the priority ceilings of all semaphores currently locked by other tasks. The contributions of this research work have been applied to the benchmarking production system : FESTO system (used as running example) allowing us to validate our results.

The Section 2 presents the benchmark production system (FESTO system). We define in Section 3 the scheduling with the Priority Ceiling Protocol. The Section 4 treats the simulation of real-time tasks with the Cheddar tool. Finally, we conclude in the last section.

II. Benchmark Production System

We present the Benchmark Production System (Detailed descriptions are available in the website: <http://aut.informatik.uni-halle.de>) : FESTO available in the research laboratory at the Martin Luther University in Germany. The FESTO Benchmark Production System is a well-documented demonstrator used by many universities for research and education purposes, and it is used as a running example in the context of this paper. FESTO is composed of three units: Distribution, Test and Processing units (Figure 1). The Distribution unit is composed of a pneumatic feeder and a converter to forward cylindrical work pieces from a stack to the testing unit which is composed of the detector, the tester and the elevator. This unit performs checks on work pieces for height, material type and color. Work pieces that successfully pass this check are forwarded to the rotating disk of the Processing unit, where the drilling of the work piece is performed. We assume in this research work two drilling machines Drill_machine1 and Drill_machine2 to drill pieces. The result of the drilling operation is next checked by the checking machine and the work piece is forwarded to another mechanical unit.

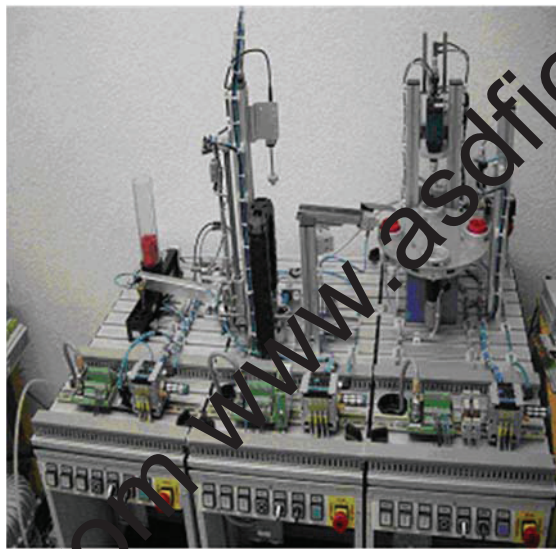


Figure 1. The FESTO system

III. Task Scheduling With Priority Ceiling Protocol

How to schedule periodic tasks with precedence and mutual exclusion constraints is considered as important as how to represent a task in a general real-time operating system. In our context, we choose the priority-driven preemptive scheduling used in the most real-time operating systems. The semaphore solution can lead to the problem of priority inversion which consists that a high priority task can be blocked by a lower priority task. To avoid such problem, we propose to apply the priority inheritance protocol proposed by (Sha, 1990).

The priority inheritance protocol can be used to schedule a set of periodic tasks having exclusive access to common resources protected by semaphores. To do so, each semaphore is assigned a priority ceiling which is equal to the highest priority task using this semaphore. A task Γ_i is allowed to enter its critical section only if its assigned priority is higher than the priority ceilings of all semaphores currently locked by tasks other than Γ_i .

Schedulability test for the priority ceiling protocol: a set of n periodic tasks using the priority ceiling protocol can be scheduled by the rate-monotonic algorithm if the following inequalities hold,

$$\forall i, 1 \leq i \leq n, C_1/T_1 + C_2/T_2 + \dots + C_i/T_i + B_i/T_i \leq i(2^{1/i}-1)$$

where B_i denotes the worst-case blocking time of a task Γ_i by lower priority tasks.

Running Example. In the FESTO Benchmark Production System, we consider three tasks R_1 (a reconfiguration task), S_1 and S_2 (service tasks) having as priority p_1, p_2 and p_3 such that $p_1 > p_2 > p_3$.

The sequence of processing steps for each task is as defined in the section previous paragraph where S (resp. R) denotes the service (resp. reconfiguration) semaphore:

$R_1 = \{ \dots P(R) \text{ execute reconfiguration } V(R) \dots \}$

$S_1 = \{ \dots P(S) \dots P(R) \dots V(S) \text{ execute service } P(S) \dots V(R) \dots V(S) \dots \}$

$S_2 = \{ \dots P(S) \dots P(R) \dots V(S) \text{ execute service } P(S) \dots V(R) \dots V(S) \dots \}$

Therefore, the priority ceiling of the semaphore R is equal to the task R_1 (because the semaphore R is used by the tasks R_1, S_1 and S_2 and we know that the task R_1 is the highest priority) and the priority ceiling of the semaphore S is equal to the task S_1 (because the semaphore S is used by the tasks S_1 and S_2 and the priority task of S_1 is higher).

We suppose that the task S_2 is running when the task S_1 is created at the instant t_3 . We suppose also that the task R_1 is created at the instant t_5 .

In the Figure 2, a line in a high level indicates that the task is executing, a line in a low level indicates that the task is blocked or preempted by another task.

The following Table 2 explains more in details the example.

TABLE I. The event and its corresponding action in the Figure 2

Event	Action
t_0	S_2 begins execution.
t_1	S_2 locks S . The task S_2 inherits the priority of S .
t_2	The task S_1 is created. As it has more priority than S_2 , it begins its execution.
t_3	The task S_1 fails to lock S as its priority is not higher than the priority ceiling of the locked S . The task S_2 resumes the execution with the inherited priority of S .
t_4	The task S_2 locks R . The task S_2 inherits the priority of R . The task R_1 is created and preempts the execution of S_2 as it has the highest priority).
t_5	The task R_1 fails to lock R as its priority is not higher than the priority ceiling of the locked R . The task S_2 resumes the execution of the critical section.
t_6	The task S_2 unlocks S .
t_7	The task S_2 executes a service.
t_8	The task S_2 locks S .
t_9	The task S_2 unlocks R and therefore has as priority the same as S . The task R_1 becomes having the highest priority. As it has more priority than S_2 , it resumes its execution.
t_{10}	The task R_1 locks R .
t_{11}	The task R_1 executes the reconfiguration.
t_{12}	The task R_1 unlocks R .
t_{13}	The task R_1 terminates its execution.
t_{14}	The task S_2 unlocks S (thus S_2 becomes having the lowest priority). Therefore, the task S_1

	resumes its execution.
t15	The task S1 locks S.
t16	The task S1 locks R.
t17	The task S1 unlocks S.
t18	The task S1 executes its service.
t19	The task S1 locks S.
t20	The task S1 unlocks R.
t21	The task S1 unlocks S.
t22	The task S1 achieves its execution.
t23	The task S2 resumes the execution and terminates its service.

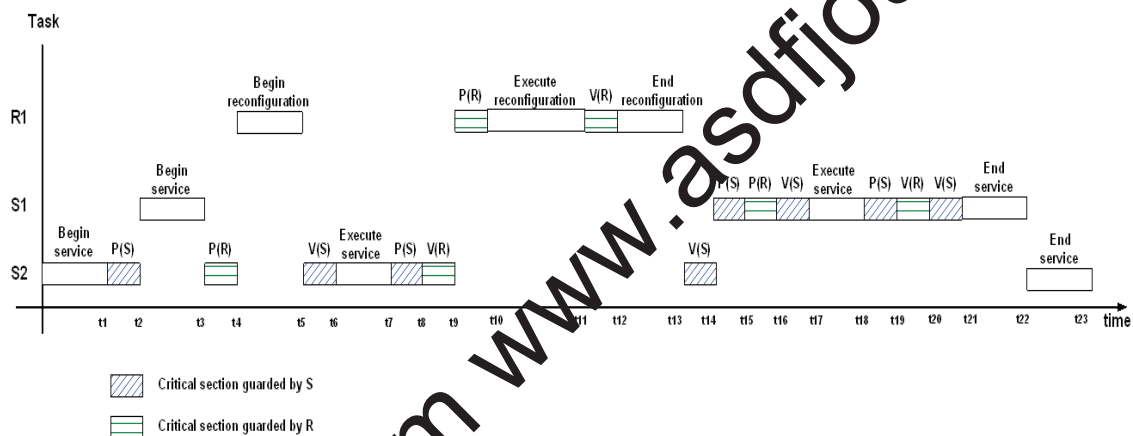


Figure 2. The priority ceiling protocol applied to three tasks R1, S1 and S2

IV. Simulation With The Cheddar Tool

Cheddar, developed at the University of Brest, is a free real-time scheduling tool (<http://beru.univ-brest.fr/~singhoff/cheddar/>). It is used to check temporal properties on real-time systems or applications. The tool provides a simulation engine and feasibility tests (Figure 3).

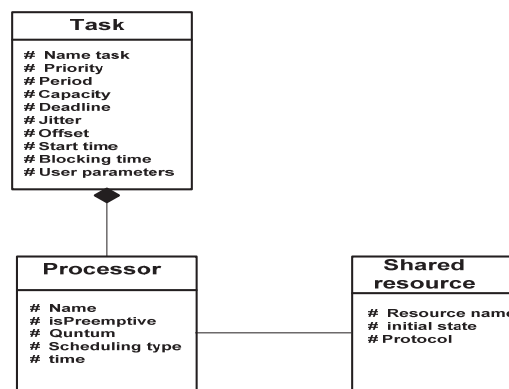


Figure 3. Cheddar Architecture

The use of Cheddar is very simple through its graphical interface.

First of all, a processor must be defined via the Edit/Update processors. To do so, a name, a scheduler policy and the nature (it is preemptive or not) must be indicated.

Secondly, a memory address is precised in the Edit/Update address spaces by specifying its name.

Finally, all the tasks have to be defined in the Edit/Update tasks by indicating its characteristics.

Scheduling with Cheddar

- ✓ Definition of the processor: We aim to simulate the Priority Ceiling Protocol with the Cheddar tool. For that reason, we define a processor by giving him a name (*Processor1*), and choosing as scheduler "POSIX 1003.1b/Highest Priority First". We choose a preemptive scheduling which is RTEMS_PREEMPT (Figure 4).

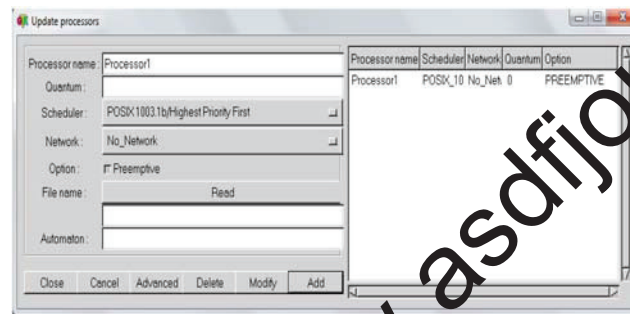


Figure 4. Definition of a processor

We add also a space of addressing associated with the processor by indicating its name and the processor to which it is associated.

- ✓ Definition of the tasks: It is necessary to define the tasks and their properties. There are 4 types of tasks, periodic, aperiodic, fish or parametric. The kind of task which interest us here is periodic. Afterthat, it is necessary to choose between policies SCHED_FIFO or SCHED_RR used for the scheduling of the same task priority. In this case, we choose SCHED_FIFO. It is also necessary to define the priority of the task. The priorities varies from 1 to 255, highest being 255. Lastly, it is necessary to specify work, starting times, deadline, and the period. These parameters are all obligatorily. In our example we create three tasks (T_1 , T_2 and T_3) having as priority respectively (1, 2 and 1) (Figure 5).

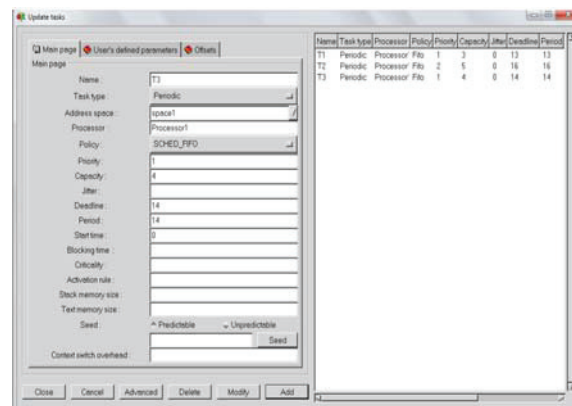


Figure 5. Definition of a task with Cheddar

- ✓ Definition of the resource: As we need semaphores, resources should be added. We indicate the name of the resource, his initial value, and the intervals during which a task wants to reach the resource. These intervals cannot be null, i.e. a semaphore cannot be taken then delivered immediately. It is also necessary to specify the resource sharing protocol. Under Cheddar, the protocols available were No Protocol, PIP, PCP or IPCP.

In our example, we define a resource (*resource1*) associated with the PCP. The task T_1 needs to use this resource during the interval [2,3] whereas the tasks T_2 wants to use it during the interval [2,5] (Figure 6).

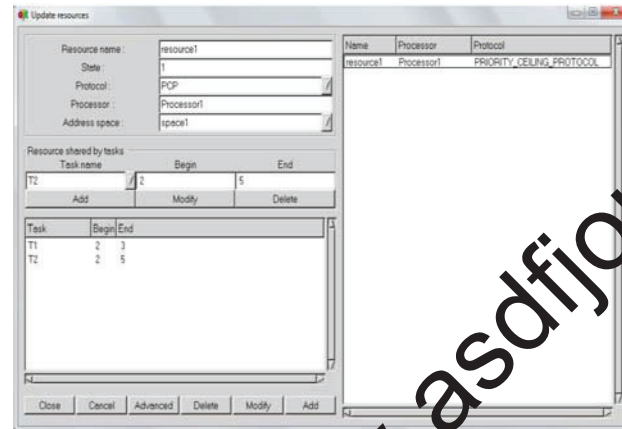


Figure 6. Definition of a resource

The Figure 7 depicts an analysis of this test case by Cheddar. The top part of the window shows the thread scheduling. The worst case response time computed from this scheduling and with feasibility tests are shown in the bottom part of the window.

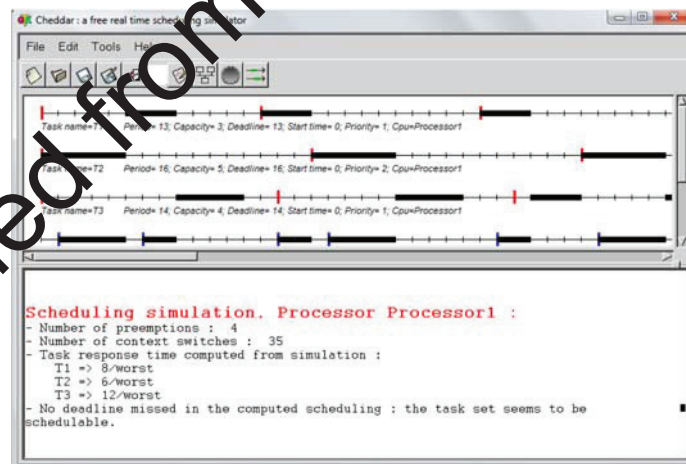


Figure 7. Scheduling with the Cheddar tool

V. Conclusion

This paper has as objective the study of Real-Time Task for Embedded Control System. We use the priority ceiling protocol as a method to ensure the scheduling between periodic tasks with precedence and mutual exclusion constraints.

The novelty of this paper is the study of dynamic reconfiguration with semaphore ensuring the following points: (i) blocking connections without blocking involved components; (ii) safety and correctness of the proposed solution; (iii) independence of any specific language; (iv) verification of consistency (i.e. logical constraints);(v) suitable for large-scale applications.

In the future work, we will be interested in the problem of allocation of real-time tasks to the devices of the execution environment and the assignment of these instances to the operating systems.

References

- [1] Merijn de Jonge, "Developing Product Lines with Third-Party Components", Electronic Notes in Theoretical Computer Science, pages 63-80, 2009.
- [2] David B. Stewart, Richard A. Volpe and Pradeep K. Khosla, "Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects", IEEE Transactions on Software Engineering (23), pages 592-600, 1997.
- [3] Roel Wuyts, Stephane Ducasse and Oscar Nierstrasz, "A data-centric approach to composing embedded, real-time software components", The Journal of Systems and Software (74), pages 25-34, 2005.
- [4] Ed Brinksma and all, "ROADMAP: Component-based Design and Integration Platforms", <http://www.artist-embedded.org>, 2003.
- [5] L. Sha, R. Rajkumar and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Trans. on Computers. Vol. 39(9) (pp. 1175-1185), 1990.